Creating I/O Kit Drivers for Multiple Architectures and OS Versions

Session 502

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Creating I/O Kit Drivers for Multiple Architectures and OS Versions

Ethan Bold
I/O Kit Engineer

Jay Towslee
I/O Kit Engineer

Introduction

- Not an introduction to I/O Kit
- Build a single I/O Kit kernel extension that runs "everywhere"…
  - Runs across kernel architectures
  - Talks to user processes of different architectures
    - Handles 64-bit address spaces

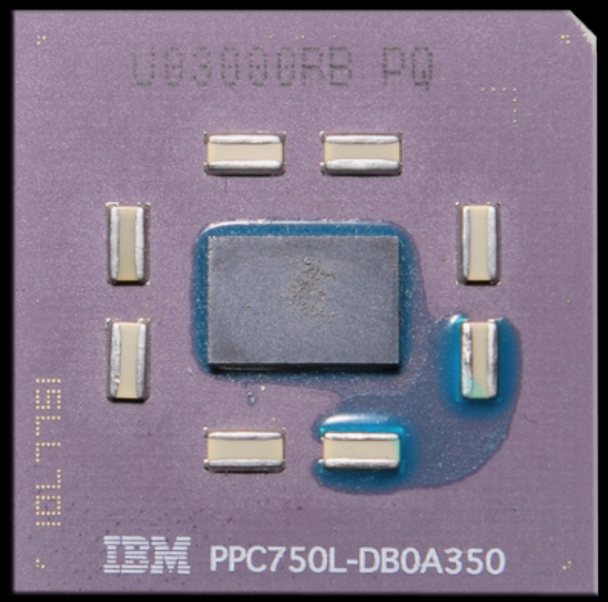
What You'll Learn

- Kernel architectures
- User process architectures
- K64
- Demo

Building Drivers for Multiple Kernel Architectures

In the Beginning

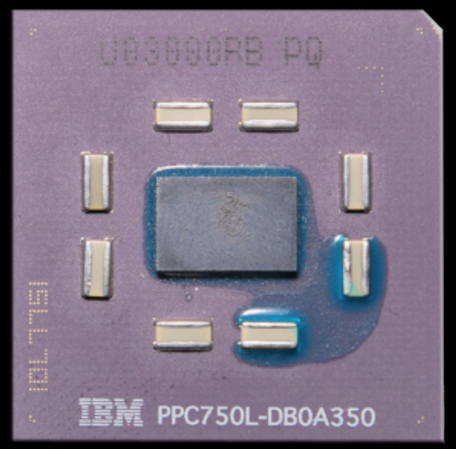
Supported kernel architecture 10.0–10.4



PPC

Intel Invasion

Supported kernel architectures 10.4.4–10.5



PPC

i386

Intel Invasion 64-Bit

Supported kernel architectures in 10.6

i386

x86_64

AvailabilityMacros.h

- Build time constraints on what Mac OS X versions the resulting KEXT will be run

```
MAC_OS_X_VERSION_MIN_REQUIRED
MAC_OS_X_VERSION_MAX_ALLOWED
```

```
/System/Library/Frameworks/Kernel.Frameworks/AvailabilityMacros.h
```

- For 10.4:

```
#include "AppleSamplePCIAvailability.h"
```

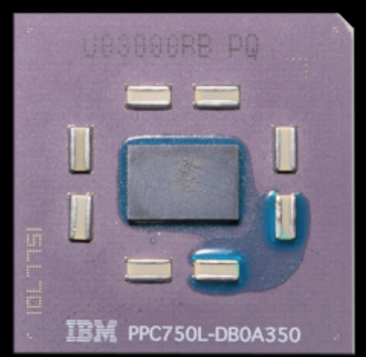
  - From AppleSamplePCI project

IOUserClient with Preprocessor Macros

```
#if MAC_OS_X_VERSION_MIN_REQUIRED <= MAC_OS_X_VERSION_10_4
   /*
    * Look up the external methods - supply a description of the parameters
    * available to be called
    *
    * This is the old way which only supports 32-bit user processes.
    */
   IOExternalMethod* SamplePCIUserClientClassName::getTargetAndMethodForIndex(
               IOService** targetP, UInt32 index)
   { }
#else
   /*
    * This is the new way which supports 32/64-bit user processes.
    */
   IOReturn SamplePCIUserClientClassName::externalMethod(
               uint32_t selector, IOExternalMethodArguments* arguments,
               IOExternalMethodDispatch * dispatch, OSObject * target, void * reference )
   { }
#endif
```

API Changes in Universal SDK (10.4u)

- Supports ppc and i386
- Focus: Adding i386 support
- Rosetta
- IODMACommand



PPC

i386

API Changes in 10.5 SDK

- Supports ppc and i386
- Focus: Adding 64-bit support
- IOUserClient moves to 64-bit API
  - For 10.4 and earlier, implement `getTargetAndMethodForIndex()`
  - For 10.5 and later, implement `externalMethod()`
- IOMemoryDescriptor moves to 64-bit API

x86_64

IOUserClient with Preprocessor Macros

Reprise

```
#if MAC_OS_X_VERSION_MIN_REQUIRED <= MAC_OS_X_VERSION_10_4
   /*
    * Look up the external methods - supply a description of the parameters
    * available to be called
    *
    * This is the old way which only supports 32-bit user processes.
    */
   IOExternalMethod* SamplePCIUserClientClassName::getTargetAndMethodForIndex(
               IOService** targetP, UInt32 index)
   { }
#else
   /*
    * This is the new way which supports 32/64-bit user processes.
    */
   IOReturn SamplePCIUserClientClassName::externalMethod(
               uint32_t selector, IOExternalMethodArguments* arguments,
               IOExternalMethodDispatch * dispatch, OSObject * target, void * reference )
   { }
#endif
```

10.5 SDK IOMemoryDescriptor

Deprecated	In favor of
IOMemoryDescriptor::getPhysicalSegment64()	IOMemoryDescriptor::getPhysicalSegment(kIOMemoryMapperNone)
IOMemoryDescriptor::getVirtualSegment()	IOMemoryDescriptor::map()->getVirtualAddress()
IOMemoryDescriptor::initWithAddress(void *)	IOMemoryDescriptor::initWithOptions()
IOMemoryDescriptor::initWithAddress(task_t)	IOMemoryDescriptor::initWithOptions()
IOMemoryDescriptor::initWithPhysicalAddress()	IOMemoryDescriptor::initWithOptions()
IOMemoryDescriptor::initWithPhysicalRanges()	IOMemoryDescriptor::initWithOptions()
IOMemoryDescriptor::initWithRanges()	IOMemoryDescriptor::initWithOptions()
IOMemoryDescriptor::map(task_t)	IOMemoryDescriptor::createMappingInTask()
IOMemoryDescriptor::withAddress(task_t)	IOMemoryDescriptor::withAddressRange()
IOMemoryDescriptor::withPhysicalRanges()	IOMemoryDescriptor::withOptions(kIOMemoryTypePhysical)
IOMemoryDescriptor::withRanges()	IOMemoryDescriptor::withOptions(kIOMemoryTypeVirtual)
IOMemoryDescriptor::withSubRange()	IOSubMemoryDescriptor::withSubRange([kIOMemoryThreadSafe])
IOBufferMemoryDescriptor::initWithBytes()	IOBufferMemoryDescriptor::initWithPhysicalMask()
IOBufferMemoryDescriptor::initWithOptions(void *)	IOBufferMemoryDescriptor::initWithPhysicalMask()
IOBufferMemoryDescriptor::initWithOptions(task_t)	IOBufferMemoryDescriptor::initWithPhysicalMask()

IOMemoryDescriptor

```
#if defined(__ppc__) && (MAC_OS_X_VERSION_MIN_REQUIRED <= MAC_OS_X_VERSION_10_4)
      // construct a memory descriptor for the out of line client memory
      // old 32 bit API - this will fail and log a backtrace if the task is 64 bit
      memDesc = IOMemoryDescriptor::withAddress( (vm_address_t) clientAddr, (
                IOByteCount) size, kIODirectionNone, fTask );
#else
      // 64 bit API - works on all tasks, whether 64 bit or 32 bit
      memDesc = IOMemoryDescriptor::withAddressRange( clientAddr, size, kIODirectionNone, fTask );
#endif
```

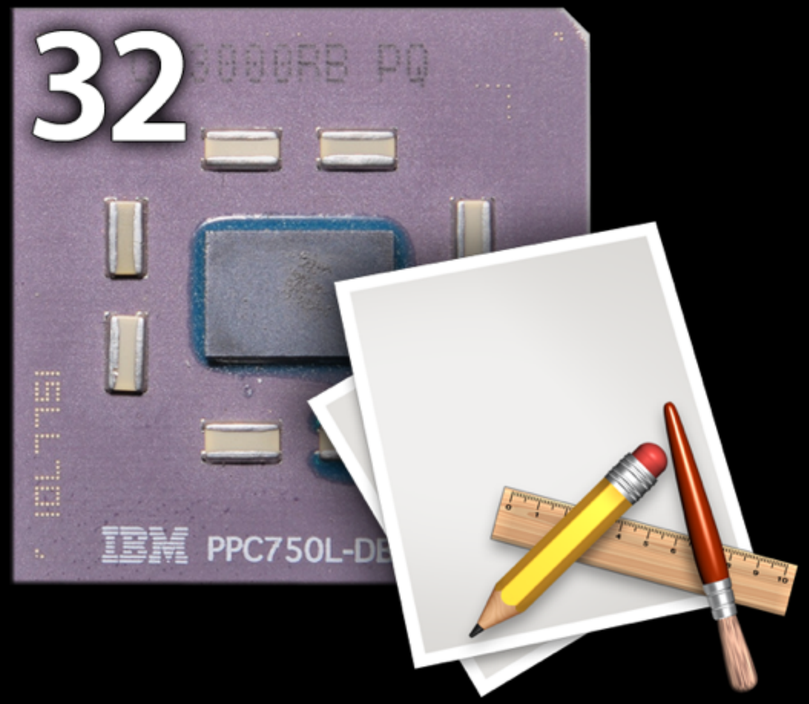
User Process Architectures

Communicating with Processes

- Drivers execute inside the kernel
- Processes execute in their own (non-kernel) tasks
- Frequently, drivers need to communicate with processes

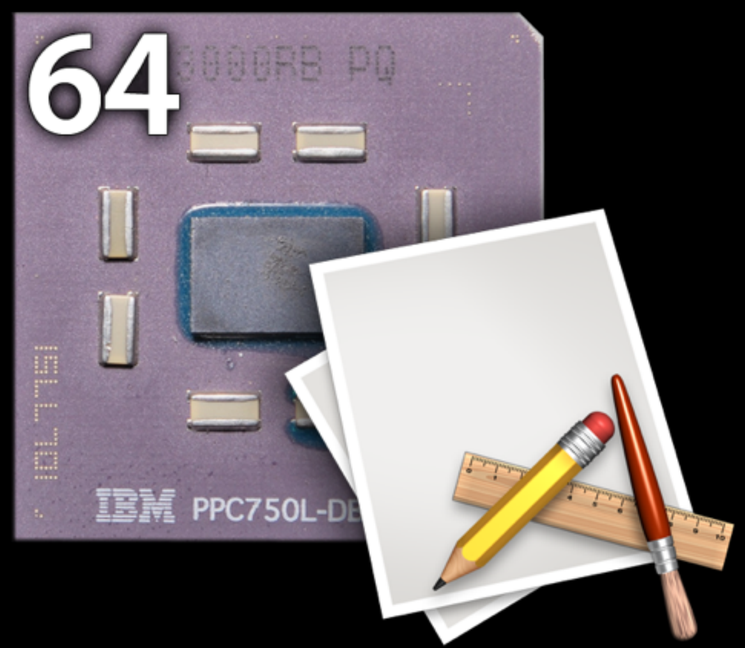
In the Beginning

All apps were PPC until 10.3.9



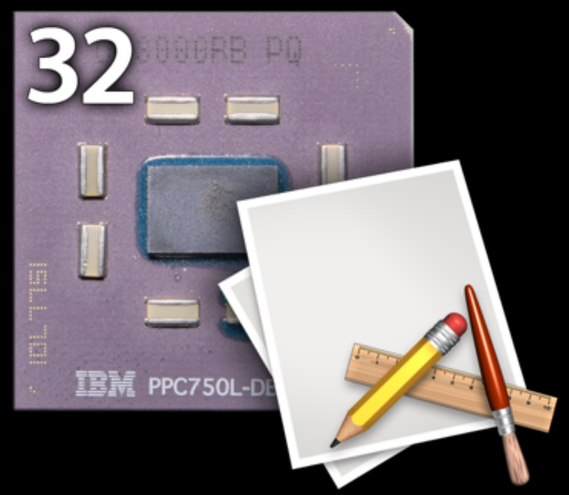
Brief PPC64 Support

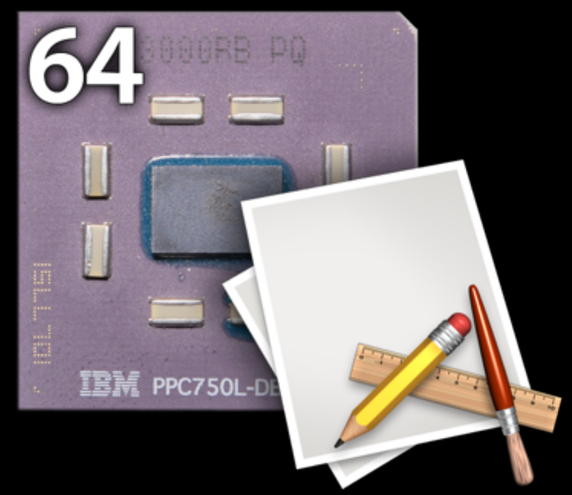
64-bit PPC user space in 10.4–10.5

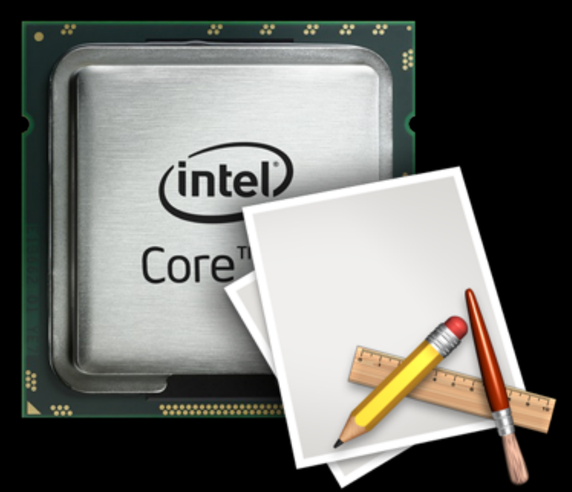


Universal Apps Arrive

Four-way starting in 10.4.7







Bridging the Divide

- If your device doesn’t communicate with user space at all
  - You are done!
- If your device doesn’t provide custom access
  (beyond what the appropriate I/O Kit family provides)
  - Let your family handle user issues
- Otherwise…

I/O Kit Provides Two Bridges

- The IORegistry
  - Trivial to get properties and structured data
  - Properties can be set with simple driver cooperation
  - **Free:** Byte order and addressing handled for free by I/O Kit

- IOUserClient subclasses
  - Preferred for any substantial amount of data
  - Provides much more flexibility and control
  - **Work required:** You must explicitly handle byte order and addressing in your driver

IORegistry

- Userland

```
IORegistryEntrySetCFProperties(service, dictRef);
```

- KEXT

```
IOReturn SamplePCIClassName::setProperties(OSObject* properties)
{
    OSDictionary* dict;
    OSNumber* number;

   dict = OSDynamicCast(OSDictionary, properties);
   if (dict) {
       number = dict->getObject(kMyDisplayValueKey);
    }
}
```

IORegistry

Modifying a collection

- KEXT

```
OSDictionary* dict = OSDynamicCast(OSDictionary, copyProperty(kMyDisplayParametersKey));
OSDictionary* copyDict = (OSDictionary *) dict->copyCollection();
if (copyDict != NULL) {
   OSDictionary* copyBrightnessDict =
      OSDynamicCast(OSDictionary, copyDict->getObject(kMyDisplayBrightnessKey));
   if (copyBrightnessDict != NULL) {
      OSNumber* num =
         OSDynamicCast(OSNumber, copyBrightnessDict->getObject(kMyDisplayValueKey));
      if (num != NULL) {
         num->setValue(value);
         // setProperty correctly serializes I/O Registry updates for our protection.
         setProperty(kMyDisplayParametersKey, copyDict);
      }
   }
   copyDict->release();
}
```

IOUserClient

Calling from user space

```
#if MAC_OS_X_VERSION_MIN_REQUIRED <= MAC_OS_X_VERSION_10_4
    kr = IOConnectMethodStructureIStructureO( connect, kSampleMethod1,
                        sizeof(varStructParam),  /* structureInputSize */
                        &structureOutputSize,  /* structureOutputSize */
                        &varStructParam,        /* inputStructure */
                        &varStructParam);       /* ouputStructure */
#else
    kr = IOConnectCallStructMethod( connect, kSampleMethod1,
                        // inputStructure
                        &varStructParam, sizeof(varStructParam),
                        // ouputStructure
                        &varStructParam, &structureOutputSize );
#endif
```

IOUserClient

Sharing data from KEXT

- Register Rosetta capability

```
// This code will do the right thing on both PowerPC- and Intel-based systems because the cross-endian
// property will never be set on PowerPC-based Macs.
fCrossEndian = false;
if (getProperty(kIOUserClientCrossEndianKey))
{
   // A connection to this user client is being opened by a user process running using Rosetta
   if (setProperty(kIOUserClientCrossEndianCompatibleKey, kOSBooleanTrue))
   {
      fCrossEndian = true;
   }
}
```

- Support Rosetta

```
structOut->results1 = 0x87654321;
// Rosetta
if (fCrossEndian)
{
   structOut->results1 = OSSwapInt64(structOut->results1);
}
```

K64

K64

- LP64
  - 0x7fffffff…
- Structure alignment
- mach_vm_address_t and careful pointer handling
- API
  - IOUserClient::externalMethod
  - IOMemoryDescriptor

mach_vm_address_t

- Kernel

```
#if MAC_OS_X_VERSION_MIN_REQUIRED <= MAC_OS_X_VERSION_10_4
  IOVirtualAddress address32 = memMap->getVirtualAddress();
#else
  mach_vm_address_t address64 = memMap->getAddress();
#endif
```

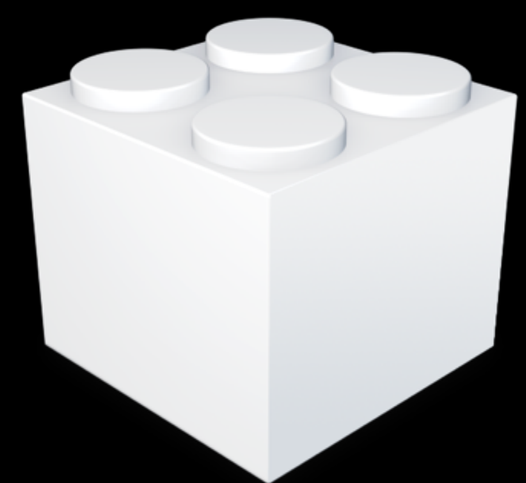
- Userland

```
#if __LP64__
    mach_vm_address_t   addr;
    mach_vm_size_t      size;
#else
    vm_address_t      addr;
    vm_size_t       size;
#endif
```

Over to Jay

K64 Development Helpers

- Move to Snow Leopard
  - gcc-4.2 as system compiler
  - Replace calls to deprecated API and classes
- Enable (and resolve) compiler warnings
  - `-Wall`
  - `-Wshorten-64-to-32`
- Architecture specific SDK
- Universal KEXTs



Demo

Summary

Ideas to take home

- Adopt new API
- Target appropriate SDK in your code
  - AvailabilityMacros.h
- Set SDK and Deployment target in Xcode
- Test and ship

More Information

Craig Keithley
I/O Technology Evangelist
keithley@apple.com

Documentation
AppleSamplePCI
http://developer.apple.com/snowleopard/library/samplecode/AppleSamplePCI

SimpleUserClient
http://developer.apple.com/samplecode/SimpleUserClient

64-Bit Transition Guide
http://developer.apple.com/documentation/Darwin/Conceptual/64BitPorting

Universal Binary Programming Guidelines
http://developer.apple.com/documentation/MacOSX/Conceptual/universal_binary

Related Sessions

Managing Kernel Extensions	North Beach Tuesday 10:30AM

Labs

64-Bit Kernel Lab	Snow Leopard Lab Wednesday 12:00PM
USB and FireWire Lab	OS X Lab B Wednesday 3:30PM
USB and FireWire Lab	OS X Lab D Thursday 12:00PM
USB and FireWire Lab	OS X Lab B Friday 9:00AM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09